Scalable Hadoop-Based Infrastructure for Big Data Analytics

Irina Astrova[1], Arne Koschel[2], Felix Heine[2], and Ahto Kalja[1(✉)]

[1] Department of Software Science, School of IT, Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia
{irina,ahto}@cs.ioc.ee

[2] Faculty IV, Department of Computer Science, Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany
akoschel@acm.org, felix.heine@hs-hannover.de

Abstract. Cloud architectures are being used increasingly to support Big Data analytics by organizations that make ad hoc or routine use of the cloud in lieu of acquiring their own infrastructure. On the other hand, Hadoop has become the de-facto standard for storing and processing Big Data. It is hard to overstate how many advantages come with moving Hadoop into the cloud. The most important is scalability, meaning that the underlying infrastructure can be expanded or contracted according to the actual demand on resources. This paper presents a scalable Hadoop-based infrastructure for Big Data analytics, one that gets automatically adjusted if more computing power or storage capacity is needed. Adjustments are transparent to the users – the users seem to have nearly unlimited computation and storage resources.

Keywords: Big Data · Cloud computing · Hadoop

1 Introduction

Big Data are big in two different senses. They are big in the quantity and variety of data that are available to be stored and processed. They are also big in the scale of analysis (or analytics) that can be applied to those data. Both kinds of "big" depend on the existence of supportive infrastructure. Such an infrastructure is increasingly being provided by the cloud like OpenStack or Amazon EC2.

The DC4C (Data Cloud for Cities) project was initiated at Hannover University of Applied Sciences and Arts. The primary goal of the DC4C project was to create a cloud-based infrastructure for Big Data analytics. An initial step toward this goal was to build a high-level architecture for such an infrastructure. A next step forward was to implement a low-level architecture based on Hadoop clusters running in the cloud.

© Springer Nature Switzerland AG 2018
A. Lupeikiene et al. (Eds.): DB&IS 2018, CCIS 838, pp. 233–242, 2018.
https://doi.org/10.1007/978-3-319-97571-9_19

2 Motivation

A Hadoop [1] cluster or more specifically HDFS (Hadoop Distributed File System) cluster consists of a *NameNode* to store the HDFS metadata and an arbitrary number of *DataNodes* that store data. Data are split into blocks and then replicated at multiple *DataNodes* to ensure high availability and performance. For data processing, Hadoop employs the MapReduce programming model, where a master node coordinates a number of *WorkerNodes* that do the actual processing.

The virtualization of a Hadoop cluster has many benefits. If a Hadoop cluster runs in the cloud, the physical hardware can be shared with other components like a web server or a relational database server to utilize the full performance of the hardware. Another advantage of a virtualized Hadoop cluster is that the setup of the cluster can be simplified by creating templates of the virtual machines, which can be easily started and stopped, thus avoiding the need of a time-intensive installation routine. This comes in conjunction with yet another benefit – a virtualized Hadoop cluster can be rapidly expanded or contracted depending on the current demand on resources.

Furthermore, all general benefits of the cloud can be applied to a virtualized Hadoop cluster like the high-availability of virtual machines or the ability to clone images for using them as a backup or node instances. On the other hand side, Hadoop is designed to offer things like reliability and high-availability by default but other things like the full utilization of the hardware by sharing them with other components bring a real benefit. Finally, not only the hardware could be shared with different components but also with another Hadoop distribution. The benefits are quite appealing but it also brings some challenges, which need to be addressed.

3 Challenges

The cloud computing technology is based on the concept of virtualization. However, the virtualization of a Hadoop cluster is a challenging task. Unlike most common server applications, Hadoop has some special requirements for a cloud architecture. In particular, Hadoop requires for topology information about the utilized infrastructure. Hadoop then uses this information to manage replications in HDFS. If the infrastructure consists of more than one cluster, Hadoop ensures that at least one replication is stored in a different hardware cluster than the other replications to allow for data access even when the whole cluster is unavailable. Moreover, Hadoop tries to perform computing tasks near the required data to avoid network transfers, which are often slower than local data access.

A cloud architecture abstracts the physical hardware to hide the infrastructure details from the hosted instances. Furthermore, shared storage pools are often used to store instances instead of having a dedicated storage in every computing node. Shared storage pools and missing topology information of the Hadoop instances might lead to multiple HDFS replications onto the same physical storage pool. Also Hadoop's paradigm to avoid network traffic by allocating computing tasks near the data storage would be broken, since shared storage pools are often connected via a network. As a consequence,

the performance of cluster would probably be massively decreased due to unnecessary replications and increased network traffic.

4 High-Level Architecture

Figure 1 gives an overview of the high-level architecture. This architecture is multilayered – different layers allocate the different responsibilities of Big Data software. An upper layer uses a lower layer as a service.

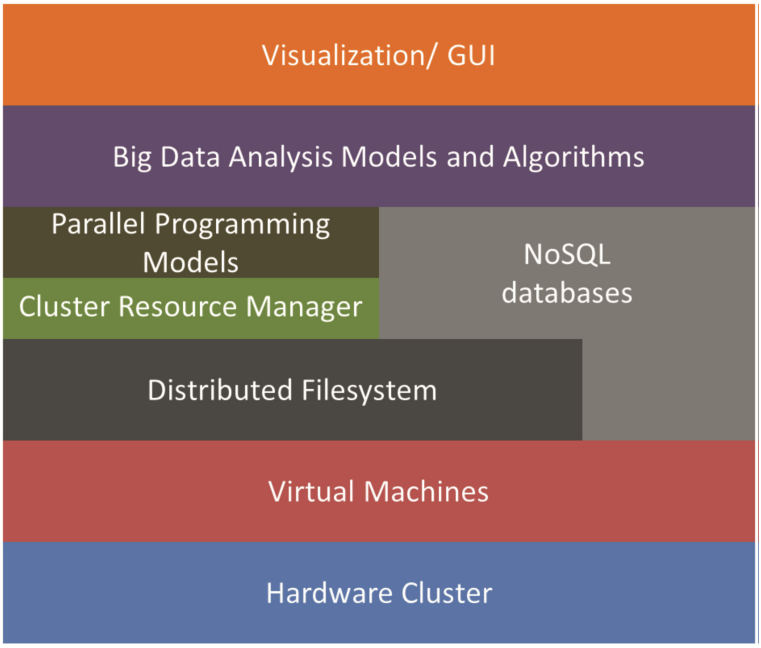

Fig. 1. High-level architecture

The architecture comprises the following layers:

- **Hardware (or physical) clusters:** This is the bottom layer. It is composed of all physical machines that are wired together either by the Internet or by a direct network connection. A physical cluster is abstracted by means of virtualization.
- **Virtual machines:** This layer is composed of all virtual machines.
- **Distributed file system:** This layer is composed of a distributed file system used for storing Big Data (typically in the range of gigabytes to terabytes) that are distributed across the virtual machines.
- **NoSQL databases:** This layer is composed of NoSQL databases like HBase. Being installed on the top of the distributed file system, NoSQL databases make it easy to create, retrieve, update and delete data using an SQL-like language. In addition, some NoSQL databases can be installed directly on the virtual machines.

- **Cluster resource manager:** This layer is responsible for managing both the physical and virtual clusters using an API (Application Programming Interface).
- **Big Data analytics models and algorithms:** This is the application layer that is responsible for Big Data analytics.
- **Parallel programming models:** The applications implement algorithms that are parallelized using programming models like MapReduce [2] and Giraph Pregel [3].
- **Visualization and GUI:** This is the top layer of the architecture. A graphical user interface (GUI) provides simple and easy access to Big Data. Furthermore, visualization improves the understanding of the results of Big Data analytics.

5 Low-Level Architecture

Figure 2 gives an overview of the low-level architecture.

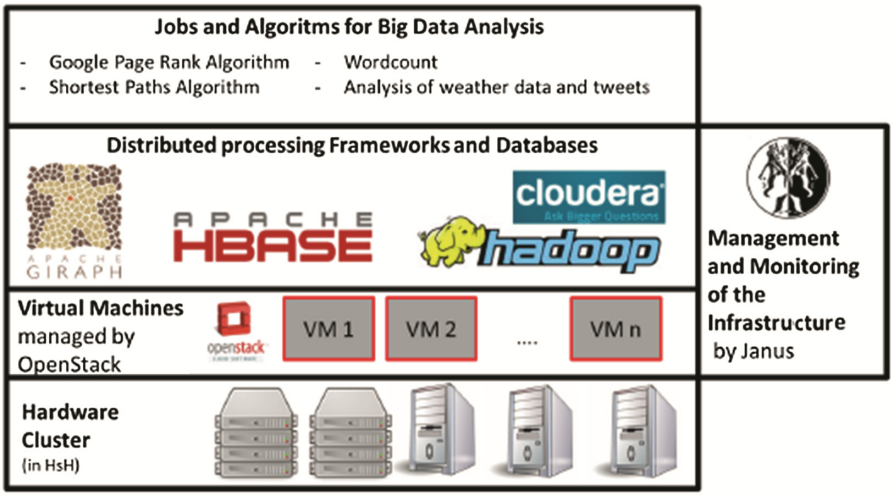

Fig. 2. Low-level architecture

The architecture comprises the following layers:

- **Hardware (or physical) clusters:** This layer is composed of five servers (Quad core Xeon, 16 GB RAM, 3 TB HDD) and three servers (Quad core Xeon, 128 GB RAM, 3 TB HDD).
- **Virtual machines:** On top of the hardware clusters, we installed virtual machines to form a hardware abstraction layer. A virtual machine acts like a physical computer except that software running on the virtual machine is separated from the underlying hardware resources. This layer is managed by OpenStack, which eases the management of virtual machines by a standardized API.

- **Distributed processing frameworks and databases:** This layer is composed of Hadoop, HDFS, HBase, MapReduce, Giraph Pregel and Cloudera. Cloudera is a distribution, which delivers many Apache products, including Hadoop and HBase.
- **Jobs and algorithms for Big Data analytics:** On this layer, we implemented many algorithms like Google's PageRank, Shortest Paths and Word Count.
- **Management and monitoring of the infrastructure:** On this layer, we implemented Janus, which monitors and tracks the virtual machines to react to storage or computing capacity bottlenecks. Janus provides an API to automate the launching, management and resizing of Hadoop clusters.

6 Implementation

It could be beneficial to co-locate the allocations of a job on the same rack (affinity constraints) to reduce network costs, but spread the allocations across machines (anti-affinity constraints) to minimize resource interference. If multiple Hadoop instances (viz., *DataNodes*) are placed on the same machine, replicated data will be-come unavailable if that machine fails.

Janus has to take care of the anti-affinity of Hadoop instances as well as to monitor both clusters (the physical cluster as well as the virtual cluster, which runs inside the physical cluster). The physical cluster is controlled by OpenStack whereas the virtual cluster is controlled by Cloudera Manager. To perform these tasks, Janus has to work with both managers. One virtual machine host should never run more than one instance from the same Hadoop cluster. This would endanger the redundancy of HDFS and might decrease the general performance with unnecessary replications. Instances forming a Hadoop cluster are anti-affine to all other instances of the same cluster but not anti-affine to instances of other Hadoop clusters. This means that one host system could generally run more than one Hadoop instance, but they cannot be from the same Hadoop cluster.

Figure 3 shows an overview of Janus architecture.

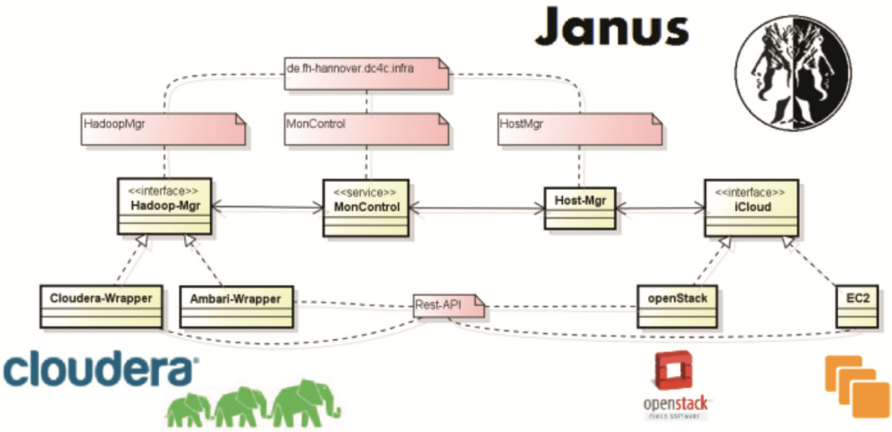

Fig. 3. Architecture of Janus [6]

The architecture mainly consists of the following components:

- **Host manager:** This component is represented by a `Host-Mgr` class with the following methods:
  - `getHadoopClusters()` returns a list of all currently used Hadoop clusters.
  - `launchHadoopCluster()` creates a new Hadoop cluster with a given number of nodes.
  - `extendHadoopCluster()` adds a given number of new nodes to the Hadoop cluster.
  - `shrinkHadoopCluster()` removes a given number of nodes.
- **Cloud manager:** This component is represented by an `iCloud` interface with the following methods:
  - `getHosts()` returns a list of all physical machines.
  - `getInstances()` returns a list of all virtual machines.
  - `createInstance()` creates a new virtual machine.
  - `deleteInstance()` deletes the virtual machine.
- **Hadoop manager** (which connects the two other managers): This component is represented by a `Hadoop-Mgr` interface with the following methods:
  - `getServices()` returns a list of all services.
  - `getServiceDetails()` returns information on a given service.
  - `addNode()` creates a new node.
  - `deleteNode()` deletes the node. This includes the decommissioning of all the services running on that node and only then the deletion of the node itself.
  - `getNodeStatus()` returns the status of a given node.

Janus is a link between the cloud manager of OpenStack and the Cloudera Manager. It connects both sides by utilizing classes, which implement two abstract interfaces: `iCloud` and `Hadoop-Mgr`. This is done to ensure that the core logic of Janus will not be altered even if cloud providers are added or removed. On the cloud management side, these are the classes of OpenStack or Amazon EC2, which implement the `iCloud` interface. On the Hadoop cluster management side, these could be the Hortonworks Ambari Manager Wrapper or the Cloudera Manager Wrapper classes, which implement the `Hadoop-Mgr` interface.

In Hadoop, there are several different roles, including *DataNode* and *WorkerNode*. A role is an instance of the service that is bound and executed on a host. As the names suggest, the *DataNode*'s role is to store incoming data, the *WorkerNode*'s role is to process the data. In most cases, these roles are combined into one node. To react dynamically to the fluctuation of the amount of incoming data, which a Hadoop cluster receives, nodes have to be created or deleted on demand. For example, when HDFS runs out of space, Janus executes the appropriate methods of the `iCloud` interface to initiate the creation of a new node with the two roles. Janus makes a synchronous call and waits for the result of the method call. If the new node is created successfully, it will execute the appropriate methods of the `Hadoop-Mgr` interface to initiate the addition of the new node to the appropriate cluster of the Cloudera Manager.

To start the roles, we used a predefined template. This template was created manually on the Cloudera Cluster setup so it can be used to create new nodes. The template was named `DC4C-Default-HDFS-MR` and included two roles: `mapreduce`, which is the *WorkerNode*, and `HDFS`, which is the *DataNode*. The template has to be called by its name and is passed to the `addNode` method of the `ClouderaManagerResource` class as a parameter. The call of the method applies the template to the newly created host and starts the roles.

To avoid the need of an additional database, Janus enforces a strict naming scheme for the Hadoop instances, which allows for the mapping only by the hostnames. Upon start up, Janus loads information about all hosts in its managed clouds via API calls to the OpenStack masters and maps the currently running instances to the hosts. Hostnames, which do not fit the naming schema, are ignored so that additional instances for other purposes can be managed manually.

7 Application Scenarios

We identified two major application scenarios for the implemented architecture. One was about the storage capacity offered to the users. If the storage capacity becomes scarce, the infrastructure will automatically increase the size of the HDFS. If the physical storage limit of the hardware gets reached, the infrastructure will automatically contact another cloud provider. This could be a private cloud provider like another university or partner organization or a public cloud provider like Amazon. The users get the possibility to define prioritization of external clouds to minimize the expenses, which arise when commercial public clouds are used. Whenever a Hadoop cluster needs to be extended, Janus searches for a new suitable host system in all managed the OpenStack clouds. Thereby currently used clouds are preferred, so that a Hadoop cluster will only be extended into a new cloud as a last resort. Within each managed cloud, Janus searches for hosts, which are currently not used by the Hadoop cluster that should be expanded. If more than one host system could run a new instance, the host with the lowest count of running instances is selected. In either case, such expansion should be considered as a temporal and rapid solution to prevent data loss, which would occur if the cloud could not store any further data. In the long term, it would be necessary to buy additional hardware to offer more storage capacity within the cloud to release expensive public cloud instances.

In the DC4C project, we had two separate OpenStack installations, each having one OpenStack master and several OpenStack computing nodes. The first cloud consisted of five servers with a quad-core processor and 16 GB RAM. The second cloud consisted of three servers with a quad-core processor and 128 GB RAM. All the servers were utilizing a local RAID-0 array as their data storage to ensure highest storage performance. Redundancy was achieved by the internal replication mechanisms of HDFS. To simulate the scaling mechanism into a public cloud, we configured Janus to treat the second cloud as the public cloud. Janus broke with the anti-affinity of instances in a Hadoop cluster in the simulated public cloud and launched new instances wherever resources were available.

Another application scenario concerned the computation power of the cloud. As more and more different users would use the cloud for their Big Data analytics, a single job could get really slow if the virtual computing nodes reach their limits. The solution for this scenario is to start further virtual computing nodes in the cloud to take over additional analysis jobs. If the physical limits of the hardware also get reached, additional computation power will be obtained from an external cloud. An important point, which has to be taken into consideration when expanding into a public cloud, is the storage location of sensitive data. The users may want not to offer those data to a public cloud provider just because the infrastructure is running out of storage. In this case, the users are given the possibility to mark their data as sensitive so that the infrastructure can avoid the exposure of those data. To realize this scenario, the cloud solution has to move other non-sensitive data to the public cloud to free storage for the sensitive data.

Based on the application scenarios, we created two rules to react to storage or computing capacity bottlenecks. Both rules are checked in a cyclic interval. If a rule gets violated, the defined action will be started and no further checks of any other rule are done until the violation is fixed. One rule is `HdfsCapacityRule`. This rule is used to monitor the free disk space in HDFS; it creates a new Hadoop node if a given threshold is violated for a given timeframe. Another rule is `MapRedSlotsRule`. It is triggered when a given percentage of the available MapReduce slots have already been in use. When this rule is violated for a given timeframe, a new Hadoop node is created too.

8 Related Work

Cloud providers have started to offer prepackaged services that use Hadoop under the hood, but do most of the cluster management work themselves. The users simply point the services to data and provide the services with jobs to run, and the services handle the rest, delivering results back to the users. The users still pay for the resources used, as well as the use of the services, but save on all of the management work [5].

Examples of prepackaged services include:

- **Elastic MapReduce:** This is Amazon Web Services' solution for managing Hadoop clusters through an API. Data are usually stored in Amazon S3 or Amazon DynamoDB. The normal mode of operation for Elastic MapReduce is to define the parameters for a Hadoop cluster like its size, location, Hadoop version and variety of services, point to where data should be read from and written to, and define steps to run jobs. Elastic MapReduce launches a Hadoop cluster, performs the steps to generate the output data and then tears the cluster down. However, the users can leave the cluster running for further use and even resize it for greater capacity.
- **Google Cloud Dataproc:** It is similar to Elastic MapReduce, but runs within Google Cloud Platform. Data are usually stored in Google Cloud Storage.
- **HDInsight:** This is Microsoft Azure's solution, which is built on top of Hortonworks Data Platform. HDInsight works with Azure Blob Storage and Azure Data Lake Store for reading and writing data used in jobs. Ambari is included as well for cluster management through its API.

Despite their advantages like ready availability and ease of use, prepackaged services only work on the cloud providers offering them. Some organizations are worried about being "locked in" to a single cloud provider, unable to take advantage of competition between the providers. Moreover, it may not be possible to satisfy data security or tracking requirements with the services due to a lack of direct control over the resources [5].

In addition to prepackaged services, cloud providers offer Hadoop-as-a-Service (HaaS) for Big Data analytics [6]. However, HaaS offerings share the same disadvantages as prepackaged services in terms of moving further away from the open source world and jeopardizing interoperability. Moreover, since unlike to prepackaged services they are not explicitly based on Hadoop, there is a separate learning curve for them, and the effort could be wasted if they are ever discarded in favor of an application that works on Hadoop or on a different cloud provider [5].

9 Conclusion

Big Data analytics requires not just algorithms and data, but also physical platforms where the data are stored and processed. This class of infrastructure is now available through the cloud.

The DC4C project was aimed at developing a cloud-based infrastructure for Big Data analytics, which gets automatically adjusted if more computing power or storage capacity is needed. One of the main challenges in the development of such an infrastructure was the integration of Big Data software framework like Hadoop into a cloud architecture as both are designed for contrary purposes. Moreover, a Big Data software framework is usually complex and its usage requires a lot of practice, knowledge and experience.

The initial result of the DC4C project was a high-level architecture for the infrastructure. A major result was the implementation of a low-level architecture based on Hadoop clusters running in the OpenStack cloud. That architecture enabled to make Hadoop clusters virtualized and scalable on demand. Furthermore, the architecture was used to evaluate the performance of different persistence layers and computational models for processing data. More specifically, the persistence layers were evaluated by comparing the storage of data onto HDFS and HBase, whereas computational models were compared by executing the PageRank algorithm with MapReduce and Giraph Pregel [4]. Finally, the architecture was recognized as being worthy of application in the area of Estonian e-Government, which also needs to deal with Big Data analytics [7, 8].

10 Future Work

In the current version of Janus, rules monitor only a single property of Hadoop cluster and check if that property violates a certain threshold. These rules were primarily used to prove that a Hadoop cluster can be automatically adjusted when the CPU usage per node increases or the HDFS capacity gets too low. A future work could be to extend the existing rule engine to support rules, which monitor multiple properties. Another

enhancement could be to monitor complex events occurred in a Hadoop cluster, e.g., when every Friday night a weekly computation is started and from week to week the performance gets lower. For this purpose, historical measurements have to be stored and evaluated.

Acknowledgement. Irina Astrova's and Ahto Kalja's work was supported by the Estonian Ministry of Education and Research institutional research grant IUT33-13.

References

1. White, T.: Hadoop: The Definitive Guide, 3rd edn. O'Reilly Media, Sebastopol (2012)
2. Shook, A.: MapReduce Design Patterns. O'Reilly Media, Sebastopol (2013)
3. Malewicz, G., Matthew, A., Bik, A., Dehnert, J., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, New York, USA (2010)
4. Koschel, A., Heine, F., Astrova, I., Korte, F., Rossow, T., Stipkovic, S.: Efficiency experiments on Hadoop and Giraph with PageRank. In: Proceedings of 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Heraklion, Crete, Greece, pp. 328–331. IEEE (2016)
5. Havanki, B.: Moving Hadoop to the Cloud Harnessing Cloud Features and Flexibility for Hadoop Clusters. O'Reilly Media, Sebastopol (2017)
6. Astrova, I., Koschel, A., Lennart, M.H., Nahle, H.: Offering Hadoop as a cloud service. In: Proceedings of the 2016 SAI Computing Conference, London, UK, pp. 589–595. IEEE (2016)
7. Kalja, A., Reitsakas, A., Saard, N.: e-Government in Estonia: best practices. In: Anderson, T.R., Daim, T.U., Kocaoglu, D.F., Piscataway, N.J. (eds.) Technology Management: A Unifying Discipline for Melting the Boundaries. pp. 500–506. IEEE (2005)
8. Kalja, A., Robal, T., Vallner, U.: New generations of Estonian e-Government components. In: Proceedings of the 2015 PICMET, Portland, Oregon, USA, pp. 625–631. IEEE (2015)